Linear Regression and Resampling Methods
FYS-STK4155 - Project 1

Gulla Serville Torvund and Karl Henrik Fredly
(Dated: October 10, 2020)

In this project we study the linear regression methods Ordinary Least Squares (OLS), Ridge and Lasso regression. We will use them to fit polynomials to the Franke function, and then study the performance of our models using bootstrap resampling and cross-validation. Finally we will find the best method and parameters to fit terrain data of a mountain in Colorado.

We find that Ridge regression with the right regularization parameter λ outperforms OLS, due to OLS suffering from overfitting for higher order polynomials. Lasso regression did not suffer as much from overfitting as OLS, but it simply under-performed on our dataset, showing no advantage over Ridge and no advantage over OLS until a very high polynomial order.

We will find that Ridge regression performs best on both the Franke function and the terrain data due to its ability to make better use of higher order polynomials without overfitting. In particular, we find that the optimal model for fitting the terrain data is found using ridge regression with a regularization parameter $\lambda = 10^{-8}$ and a 16-th order polynomial. This model achieved a mean squared error of 76.56, while the best OLS model, which was a fit using a 10-th order polynomial, only achieved a mean square error of 101.87. We also confirmed that our optimal model reproduced the major features of the terrain in a 3D plot.

I. INTRODUCTION

Linear regression with resampling is a powerful tool for making models that describe data and make predictions. It is also an essential step towards understanding more sophisticated statistical methods. We will present and evaluate three methods for linear regression, and finally use them to find the best model to describe some terrain data.

In the methods section we will present the different regression methods we will look at, Ordinary Least Squares Regression, Ridge Regression and Lasso Regression. We will also look at how bootstrapping lets us approximate the bias and variance of our model, together with how Cross-validation offers a more robust way to evaluate model performance. Furthermore, we will look at the Franke function, which we will use to test the performance of our models before using them on the terrain data.

In the results section we will present the performance of our models found using Ordinary Least Squares, Ridge and Lasso, and how it changes with the choice of model complexity, number of data points and regularization parameters. We also present the optimal fit we found to the terrain data.

Finally, in the discussion and conclusion, we analyze our results and make sense of them.

For this project we wrote code implementing OLS and Ridge regression together with Cross-Validation. All code used to calculate the results in this report can be found here, in the form of a notebook with explanations of the code: https://github.com/KarlHenrik/FYS-STK-Gruppe/tree/master/Project1.

II. METHODS

We want to evaluate three different methods for linear regression and find the best model for fitting some terrain data. To do this we must first look at how these methods work, and how we can evaluate their performance. Before we look at the specific linear regression methods we are going to be using however, we take a more general look at linear regression.

A. Linear Regression

Linear regression tries to model the linear relationship

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

between the dependent variable $\mathbf{y}$ and the independent variable X by finding the parameter $\boldsymbol{\beta}$ which "best" satisfies the equation.

$\boldsymbol{\varepsilon}$ is the noise in our measurement of $\mathbf{y}$ which we assume to be normally distributed with variance σ^2 and expectation value 0. The noise $\boldsymbol{\varepsilon}$ makes it so that our model given by the parameters $\boldsymbol{\beta}$ won't necessarily reproduce the wanted output.

Furthermore, the equation is most likely impossible to satisfy given that there might not be a linear relationship between $\mathbf{y}$ and X, especially with the noise $\boldsymbol{\varepsilon}$. We accept this, and try to find the parameters $\boldsymbol{\beta}$ which give us the model with the best performance, the model which minimizes some chosen cost function $C(\beta)$.

B. Measurement of performance

Our linear model will not have a perfect fit, but we can evaluate how well it performs with a cost function.

Our choice of cost function decides our linear regression method.

The simplest cost function is the Mean Squared Error (MSE)

$$MSE(\mathbf{y},\tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2,$$

Where y_i is the i-th element of $\mathbf{y}$, which we want to approximate with our prediction given by our model $\tilde{\mathbf{y}} = X\boldsymbol{\beta}$, whose i-th element is $\tilde{y}_i$. The MSE measures the mean of the squares of the differences between the values in the given dataset and those predicted by our model. A low MSE indicates a good fit, as our model closely reproduces the given data.

Another measure of performance is the R^2-score given by the function

$$R^2(\mathbf{y},\tilde{\mathbf{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i-\bar{y})^2},$$

where $\bar{y}$ is the mean value of $\mathbf{y}$. If the numerator, the MSE, is close to 0, the R^2 score will be close to 1. That is, an R^2 score close to 1 indicates a good fit. We can use the R^2-score to evaluate how well a model performs, though we will not be using it as a cost function.

We will primarily use the MSE to judge how well a given model performs. An important note is that we test our model on test data, data which was not used to find the model, unless otherwise specified. We use training data to find a model by minimizing a cost function on the training data, and then we can test our model by for instance calculating the MSE on the test data. Here, we generally choose to use 20% of our data for testing.

Later on, we will see a more robust way of testing our regression methods, with bootstrapping and cross-validation.

C. Ordinary Least Squares regression (OLS)

The simplest regression method we use is the Ordinary Least Sqares method (OLS). This linear regression method simply uses the MSE as its cost function.

$$C_{OLS}(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2,$$

Assuming $\boldsymbol{X}^T\boldsymbol{X}$ is invertible, we find the optimal parameters (which minimize the cost function) $\hat{\boldsymbol{\beta}}^{OLS}$ with the equation

$$\hat{\boldsymbol{\beta}}^{OLS} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

If $\boldsymbol{X}^T\boldsymbol{X}$ nonetheless is *not* invertible, that is, if $\boldsymbol{X}^T\boldsymbol{X}$ is a singular matrix, we get a problem using the OLS method. This problem can however be solved by using the Ridge or Lasso method.

The variance of the parameters $\hat{\boldsymbol{\beta}}$ is given by $\sigma^2(\hat{\boldsymbol{\beta}}_j) = \sigma^2\sqrt{[X^TX]^{-1}}_{jj}$. We will use this to find the 95% confidence intervals $\hat{\boldsymbol{\beta}}_j \pm 1.96\sqrt{\sigma^2(\hat{\boldsymbol{\beta}}_j)}$.

D. Ridge regression

The Ridge regression method follows the OLS method closely, but will fix the potential singularity problem we can get with $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$. For this, we use an *ad hoc* approach and add a term $\boldsymbol{\lambda}I$ so that $\boldsymbol{X}^T\boldsymbol{X} \longrightarrow \boldsymbol{X}^T\boldsymbol{X}+\boldsymbol{\lambda}I$. This means that the optimal parameter $\hat{\boldsymbol{\beta}}^{Ridge}$ is given by [1]

$$\hat{\boldsymbol{\beta}}^{Ridge} = \left(\boldsymbol{X}^T\boldsymbol{X}+\boldsymbol{\lambda}I\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

A small value λ is added along the diagonal of the matrix so that we can know with certainty that it is non-singular.

The hyperparameter λ also works as a regularization parameter. It can prevent overfitting by "punishing" large regression coefficients. The hope is that by penalizing large coefficients, our regression method will find a model that tries to fit the general shape of the data and not only the training data. This will prevent that we have many large higher order terms which cancel each other out at the training data points, but lead to large errors elsewhere.

The cost function given by Ridge regression is given by

$$\begin{aligned} C_{Ridge}(\hat{\boldsymbol{\beta}}) &= \sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 + \lambda\sum_{i=0}^{n-1}(\hat{\beta}_i)^2 \\ &= ||\boldsymbol{y}-\boldsymbol{X}\hat{\boldsymbol{\beta}}||_2^2 + \lambda||\hat{\boldsymbol{\beta}}||_2^2. \end{aligned}$$

Here we see that the sum of the squares of the coefficients β_i times some normalization parameter λ is added to the MSE.

E. Lasso regression

The Lasso regression method, short for "least absolute shrinkage and selection operator" regression, is quite similar to Ridge regression. The only difference is that the penalty term added to the MSE in the cost function is the sum of the absolute value of the parameters β_i. That is [1]

$$\begin{aligned} C_{Lasso}(\hat{\boldsymbol{\beta}}) &= \sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 + \lambda\sum_{i=0}^{n-1}|\hat{\beta}_i| \\ &= ||\boldsymbol{y}-\boldsymbol{X}\hat{\boldsymbol{\beta}}||_2^2 + \lambda||\hat{\boldsymbol{\beta}}||_1. \end{aligned}$$

In practice, Ridge shrinks the parameters β_i, while Lasso cuts them down - think division vs. subtraction. This

means that with Lasso we can end up setting unimportant parameters to 0, which can be preferable. We do not have an analytical solution for the optimal parameter $\hat{\boldsymbol{\beta}}^{Lasso}$, like we did with OLS and Ridge. We will therefore use an iterative algorithm provided by SciKit-Learn to approximate the optimal parameter.

F. Bias-variance analysis with bootstrapping

In this project, we will use the bootstrap resampling technique as a means to study the bias-variance trade-off.

Before we explain bootstrapping, we split up the expression for the MSE into three terms that will represent the bias, variance and irreducible error.

$$MSE(\mathbf{y},\tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2, = \mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right]$$

We can rewrite and split up the expression for the MSE into parts that represent the bias and the variance of our approximation $\tilde{\boldsymbol{y}}$. We now write $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$, where $f(\boldsymbol{x})$ is the function which gives our target data without any noise or error. We find

$$\begin{aligned}
\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] &= \mathbb{E}\left[\boldsymbol{y}^2 - 2\boldsymbol{y}\tilde{\boldsymbol{y}} + \tilde{\boldsymbol{y}}^{\boldsymbol{2}}\right] \\
&= \mathbb{E}\left[(f(\boldsymbol{x})+\boldsymbol{\epsilon})^2 - 2(f(\boldsymbol{x})+\boldsymbol{\epsilon})\tilde{\boldsymbol{y}} + \tilde{\boldsymbol{y}}^{\boldsymbol{2}}\right] \\
&= \mathbb{E}\left[f(\boldsymbol{x})^2 + 2f(\boldsymbol{x})\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^2 - 2f(\boldsymbol{x})\tilde{\boldsymbol{y}} - 2\boldsymbol{\epsilon}\tilde{\boldsymbol{y}} + \tilde{\boldsymbol{y}}^{\boldsymbol{2}}\right]
\end{aligned}$$

We add and subtract the terms $2f(\boldsymbol{x})\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]$, $2\tilde{\boldsymbol{y}}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]$ and $2\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2$. Many of the terms will then cancel out. We assume that the noise is normally distributed with expected value 0, thus the expected value of the noise squared is equal to the variance of the noise.

$$\begin{aligned}
\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] &= \mathbb{E}\big[f(\boldsymbol{x})^2 - 2f(\boldsymbol{x})\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2 \\
&\quad + \tilde{\boldsymbol{y}}^{\boldsymbol{2}} - 2\tilde{\boldsymbol{y}}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2 + 2f(\boldsymbol{x})\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^2 \\
&\quad - 2f(\boldsymbol{x})\tilde{\boldsymbol{y}} - 2\boldsymbol{\epsilon}\tilde{\boldsymbol{y}} + 2f(\boldsymbol{x})\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + 2\tilde{\boldsymbol{y}}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] \\
&\quad - 2\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2\big] \\
&= \mathbb{E}\Big[\big(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\big)^2 + \big(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\big)^2 + \boldsymbol{\epsilon}^2 \\
&\quad + 2f(\boldsymbol{x})\boldsymbol{\epsilon} - 2f(\boldsymbol{x})\tilde{\boldsymbol{y}} - 2\boldsymbol{\epsilon}\tilde{\boldsymbol{y}} + 2f(\boldsymbol{x})\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] \\
&\quad + 2\tilde{\boldsymbol{y}}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] - 2\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2\Big] \\
&= \mathbb{E}\Big[\big(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\big)^2\Big] + \mathbb{E}\Big[\big(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\big)^2\Big] \\
&\quad + \mathbb{E}\Big[\boldsymbol{\epsilon}^2\Big] \quad + 2\mathbb{E}\left[f(\boldsymbol{x})\right]\cdot 0 + 2\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\cdot 0 \\
&\quad + 2\mathbb{E}\left[f(\boldsymbol{x})\right]\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] - 2\mathbb{E}\left[f(\boldsymbol{x})\right]\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] \\
&\quad + 2\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2 - 2\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2 \\
&= \mathbb{E}\Big[\big(f(\boldsymbol{x}) - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\big)^2\Big] + \mathbb{E}\Big[\big(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\big)^2\Big] + \sigma^2
\end{aligned}$$

From left to right we now have expressions for the bias, variance and the irreducible error.

By drawing as many random samples from the training data with replacement as there are training data, we get data which is slightly different from the original training data, but still represents random draws from real data. We can fit a model to this data and calculate the bias, variance and MSE on test data. If we do this random resampling of the training data N times (N bootstraps), and take the average bias, variance and MSE, we get a more robust measure of how a model from our regression method would perform on average. We call this bootstrapping, and we will use it to analyze the bias and variance of different models.

G. Cross-validation (CV)

Another resampling technique we will use is Cross-validation (CV), or more precisely the k-fold CV. We will resample the data by splitting it up in k equally sized subsets of data, and use k-1 subsets as training data and the remaining subset as test data. This is repeated k times so that each subset gets to be the test data one time. We average the MSE of the k tests to get an even more robust measure of the average performance of our model. We only used 10-fold CV in our calculations, and we wrote our own implementation of cross validation [2].

H. What we are fitting

In the first part of this project, we will use the mentioned regression methods for approximating a fit to the two-dimensional function called the Franke function. It is a typical choice for testing linear regression methods. The Franke function is given by [3]

$$\begin{aligned}
f(x,y) &= \frac{3}{4}\exp\left\{\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right)\right\} \\
&\quad + \frac{3}{4}\exp\left\{\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right)\right\} \\
&\quad + \frac{1}{2}\exp\left\{\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right)\right\} \\
&\quad - \frac{1}{5}\exp\left\{\left(-(9x-4)^2 - (9y-7)^2\right)\right\},
\end{aligned}$$

and is in our case defined for $x, y \in [0,1]$. Figure 1 shows the Franke function.

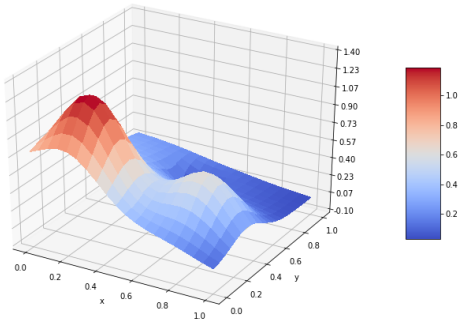

Figure 1. A 3D plot of the Franke function we will use to test our regression methods.

We will use polynomials of x and y of varying orders to fit the function. Table II shows the possible polynomial terms in a polynomial of order 5. As you can see, its shape resembles two small hills, and it will therefore hopefully work well as a testing-grounds before we try to fit real terrain data.

After evaluating the different regression methods, we will fit a polynomial to terrain data from the website https://earthexplorer.usgs.gov/. We chose to fit our polynomial to a mountain in colorado. The terrain datafile and excact coordinates can be found in our code [2]. We will use cross-validation to test several combinations of λ and polynomial order, and choose the one which performed the best to recreate the terrain. Lasso regression ended up not converging to anything useful, so we were only able to test Ridge and OLS. A plot of the terrain can be seen in figure 2. The terrain we look at consists of 40x40 datapoints.

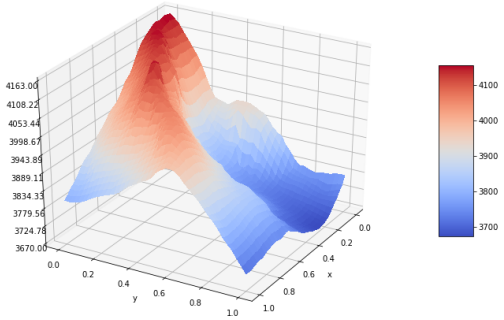

Figure 2. The terrain we will be fitting with OLS and Rigde to find the optimal model and recreate the terrain.

III. RESULTS

A. Fitting the Franke function

We first fit the Franke function with polynomials of x and y to evaluate our different regression methods. To compare the different methods on an even footing we always fit our models to 80 (different) randomly selected points on the Franke function and test our models on 20 (different) randomly selected points. The data will be scaled so that the training inputs have variance 1 and expected value 0 (except for the constant terms).

Evaluating OLS

We fit a polynomial of x and y of fifth order to the Franke function using OLS. The MSE and R2-score of our model on the training and test data is shown in table I.

Table I. The MSE and R2-score of a fifth order polynomial fit to the Franke function using OLS.

	Training Data	**Test Data**
MSE	0.0107	0.0115
R2	0.8488	0.8278

The coefficients of our polynomial fit with confidence intervals is shown in table II.

Table II. Coefficients of fifth order polynomial fit to the Franke function using OLS.

β_i **Term**	β_i **value**	β_i **Term**	β_i **value**
1	0.38 ± 0.00		
y	2.15 ± 0.10	x^2	-9.65 ± 0.36
y^2	-6.14 ± 0.44	x^2y	11.57 ± 0.38
y^3	2.72 ± 0.86	x^2y^2	-4.73 ± 0.35
y^4	4.10 ± 0.80	x^2y^3	0.62 ± 0.14
y^5	-2.94 ± 0.29	x^3	11.59 ± 0.76
x	2.47 ± 0.07	x^3y	-10.65 ± 0.35
xy	-6.10 ± 0.25	x^3y^2	1.99 ± 0.13
xy^2	9.41 ± 0.50	x^4	-4.14 ± 0.72
xy^3	-9.15 ± 0.46	x^4y	3.40 ± 0.15
xy^4	3.61 ± 0.17	x^5	-0.40 ± 0.25

We now vary the complexity of our model to see how training and test error changes with model complexity. The result is shows in figure 3.

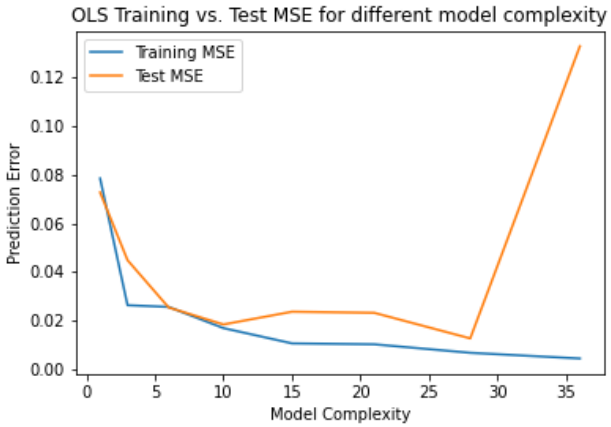

Figure 3. Training and test error of model fit to the Franke function using OLS. We see how model complexity affects the different errors.

We look to explain the bahaviour of the test MSE in figure 3 in terms of the bias-variance trade-off. Using bootstrap resampling, we calculate the bias, variance and MSE of our model shown in figure 4.

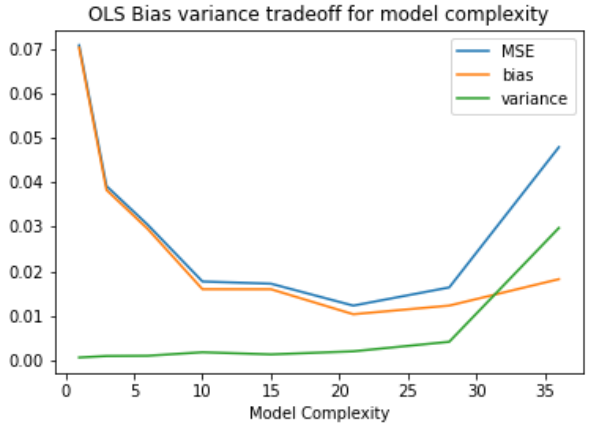

Figure 4. Bias-variance trade-off of model fit to the Franke function using OLS. We see how model complexity affects the bias, variance and MSE.

The bias, variance and MSE when increasing datapoints for our OLS model is shown in figure 5. We again used bootstrapping.

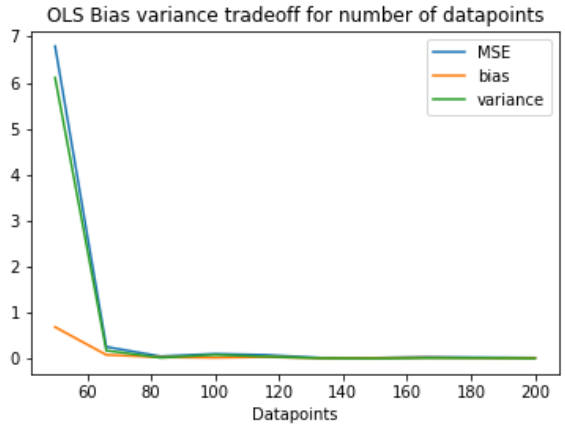

Figure 5. Bias-variance trade-off of model fit to the Franke function using OLS. We see how the number of datapoints affects the bias, variance and MSE. Used bootsrapping.

In figure 6, using cross-validation, we get a more robust measure of the MSE of OLS for different model complexities. Including the MSE from the bootstrap for comparison.

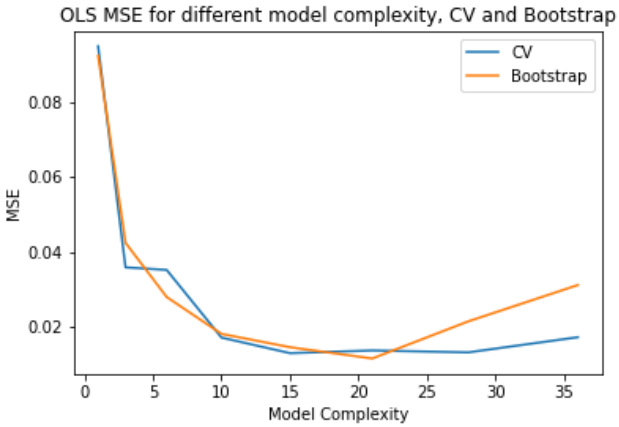

Figure 6. MSE of OLS model fit to Franke function for different model complexity. Comparing the result Bootstrapping gives with that CV gives.

Bias-variance trade-off for the regularization parameter

We calculate the bias, variance and MSE of a model fit to the Franke function using Ridge regression. We use an 8-th order polynomial, and change only the regularization parameter λ.

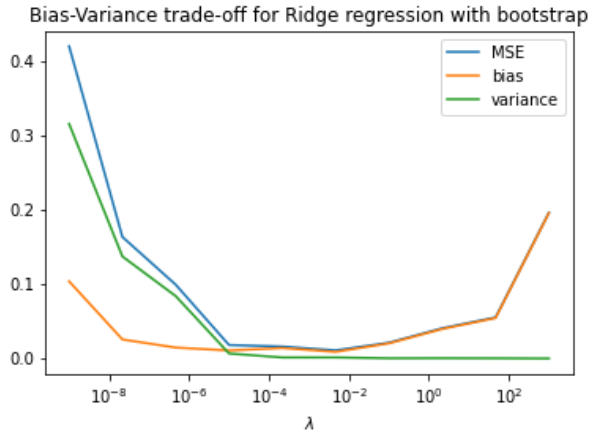

Figure 7. Bias-variance trade-off of 8-th order polynomial fit to the Franke function using Ridge regression. We see how the parameter λ affects the bias, variance and MSE.

We also calculate the bias, variance and MSE of a model fit to the Franke function using Lasso regression. We use an 8-th order polynomial, and change only the regularization parameter λ.

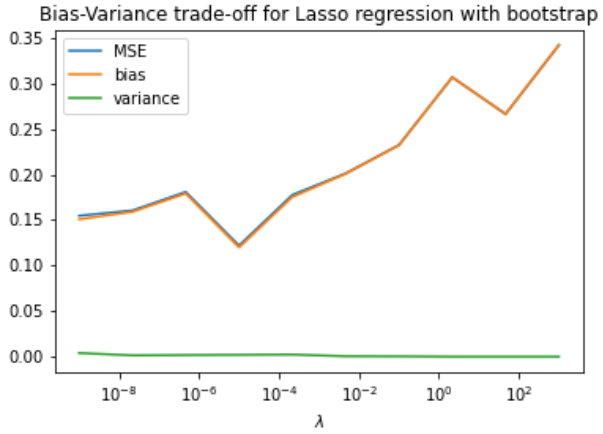

Figure 8. Bias-variance trade-off of 8-th order polynomial fit to the Franke function using Lasso regression. We see how the parameter λ affects the bias, variance and MSE.

Comparing methods

In figure 6 we see that the MSE of OLS starts to increase at about 35 parameters. We now set the polynomial order to 8, meaning we have 45 parameters. By varying the regularization parameter, we can see how Ridge and Lasso compare with OLS at this polynomial order. Result shown in 9. We use cross-validation to calculate the MSE of the methods.

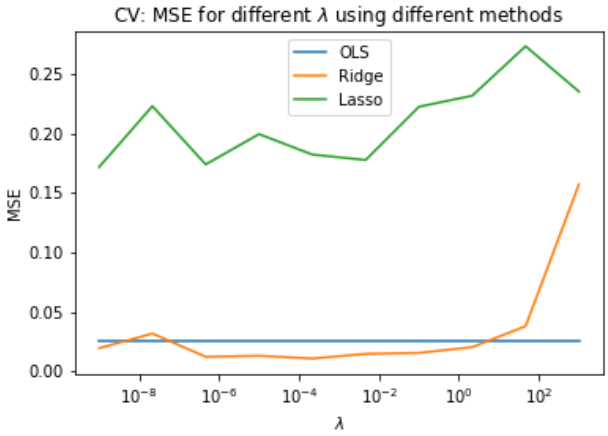

Figure 9. The MSE of OLS, Ridge and Lasso as a function of regularization parameter λ with a model complexity of 45. The models are fit to the Franke function, and are tested with cross-validation.

In figure 7 and 8 we see that the $\lambda = 10^{-5}$ works well for both Ridge and Lasso for the Franke function, so we set $\lambda = 10^{-5}$. By varying the polynomial order, we can see how Ridge and Lasso compare with OLS at different polynomial orders. We use cross-validation to calculate the MSE of the methods. Result shown in figure 10.

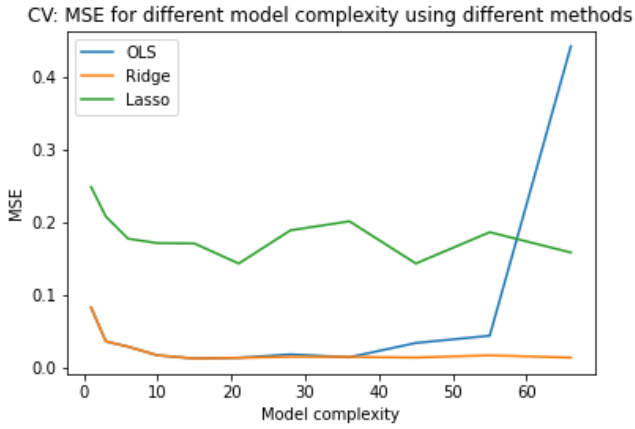

Figure 10. The MSE of OLS, Ridge and Lasso as a function of model complexity with $\lambda = 10^{-5}$. The models are fit to the Franke function, and are tested with cross-validation.

B. The Terrain data

We now look at which regression method best fits our terrain data. Lasso regression failed badly to converge, and returned only nonsensical results. The search for the best model will thus be between Ridge and OLS. Our terrain consists of 1600 evenly spaced datapoints.

A search of possible combinations of λ and polynomial order resulted in Ridge regression having the lowest MSE.

log_{10} of the MSE of some of the best combinations are shown in figure 11. These values were found using cross-validation.

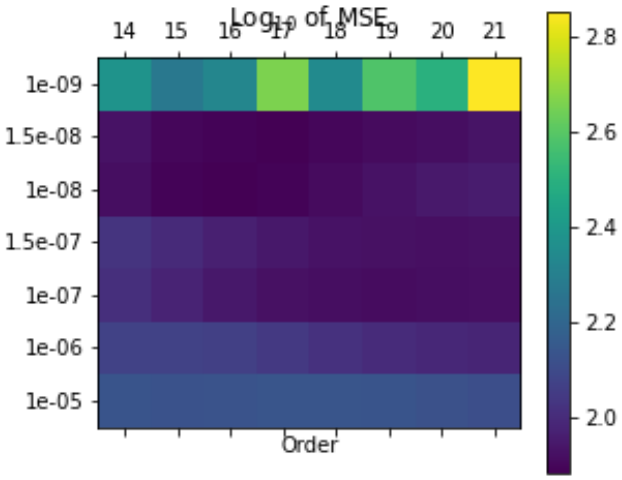

Figure 11. log_{10} of the MSE of ridge regression found using cross-validation. Testing combinations of parameters λ and polynomial order. The smallest MSE was found for the combination $\lambda = 10^{-8}$ and order = 16.

The optimal combination of parameters found were $\lambda = 10^{-8}$ and a polynomial order of 16 for Ridge Regression. Cross-validation found the MSE of that combination of parameters to be 76.56.

Cross-validation found an optimal polynomial order of 10 for OLS, with a MSE of 101.87.

By using our optimal parameters for Ridge to fit 80% of the terrain data, we were able to get the recreation of the terrain seen in figure 12.

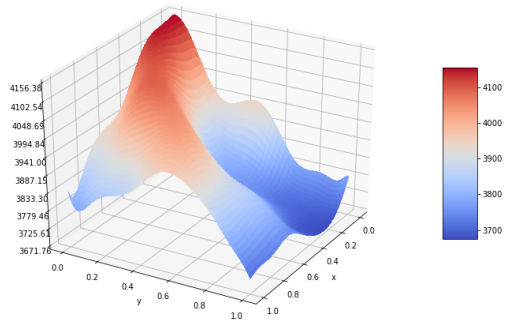

Figure 12. Terrain approximation from ridge regression model using the parameters we found to be optimal for fitting the terrain. We used 80% of our 1600 terrain datapoints to fit the model, and predicted the elevation of the terrain at 10000 evenly spaced points.

IV. DISCUSSION

Ordinary least squares regression (OLS) performed as expected. In table I we see that the MSE and R2-score on the test data is only slightly worse than on the training data. In figure 3 we see that the test error becomes much worse for higher polynomial order. This is an example of overfitting, where the model has enough degrees of freedom that when it tries to fit the training data closely, it fails to fit the general shape of the data.

Figure 4 shows this in detail. With increasing polynomial order, the bias of the model decreases as it on average has values close to the function we want to fit. But with an increasingly complex model, the model starts to vary a lot around this average, resulting in a high variance and bad test results. A way to bypass this bias-variance tradeoff is by having more data to fit your model, as shown in figure 5, where the variance of the model quickly drops with increasing data. We mostly use 100 datapoints in this project, because any more would lead to much less overfitting, and much less interesting figures. Using Cross–Validation, we see in figure 6 that the overfitting seen in figure 3 and 4 was likely exaggerated.

Another way to counteract overfitting is by penalizing large coefficients in the polynomial. Ridge regression aims to do this by shrinking the coefficients, while Lasso regression cuts them down (again, Ridge typically makes coefficients smaller by some factors, while Lasso makes them smaller by some values, often setting coefficient to 0). For very small λ, the models are very close to OLS, while for very large λ they over-penalize the coefficients, leading to nonsensical models. We see this in figure 7, where we have poor performance for small λ, due to OLS being quite poor when fitting with an 8-th degree poly-

nomial. We also have poor performance for very large λ, as expected. And it seems like the desired result is achieved, a smaller MSE for some λ in between.

Figure 8 shows that lasso performed quite poorly in general when fitting the Franke function. Lasso works well when we have a few important parameters which fit the function well, and some parameters which fit the function poorly. Lasso then removes the unimportant parameters, while keeping the important ones. However, as we see in table II, this model which fit the data quite well has many significant terms. This means that Lasso does not have the wanted effect, instead resulting in large bias as shown in figure 8

In figure 9 we see that Ridge outperforms OLS in the range of λ which also resulted in a small MSE in figure 7. Lasso still performs badly here, as expected. Figure 10 shows that with a suitable λ like 10^{-5}, Ridge prevents overfitting for large polynomial orders. Lasso also prevents this overfitting, but stil does much worse than Ridge.

Figure 11 shows the same effect as 7, where too small or too large λ results in a large MSE. Here, the best compromise between model complexity and regularization parameter was $\lambda = 10^{-8}$ and an order of 16. This is a much higher polynomial order than we looked at with the Franke function, and a much smaller best λ. The reason for this is that we have many more data points, and the data is more complex. This makes higher order polynomials perform much better. This is backed up by the fact that a polynomial of order 10 performed best for the OLS regression.

The terrain approximation in figure 12 fits quite well to the terrain in figure 2. It does not have as much detail in the sharp ridge of the mountain, and we see some tendency to spike up or down in the corners, but it does a good job of recreating the complex shape of the mountain.

Lasso not giving a usable answer for the terrain was unfortunate, but it was unlikely to outperform Ridge as the data does not resemble any simple combination of a few polynomials. Ridge outperforming OLS was expected, since we saw Ridge being able to make better use of higher order polynomials, which is essential to fitting something as complex as a real mountain.

V. CONCLUSION

We have evaluated the three regression methods Ordinary Least Squares, Ridge and Lasso regression on 3D data with bootstrap resampling and cross-validation. We also used cross-validation to find the optimal method and parameters to fit terrain data of a mountain in Colorado.

We found that OLS suffered from overfitting for higher order polynomials, which meant it could not capture the complexity of the data without suffering from high variance. Lasso regression solved the high variance problem by penalizing large coefficients in the polynomial, but ultimately ended up unable to fit the data better than even OLS due to a large bias. When fitting the terrain data, Lasso completely failed to converge on a usable solution.

We found Ridge regression to be the best method for fitting our data, due to it being able to utilize higher order polynomials without suffering from high variance or high bias. The optimal model for fitting the terrain data is found using ridge regression with a regularization parameter $\lambda = 10^{-8}$ and a 16-th order polynomial. This model achieved a mean squared error of 76.56, while the best OLS model, which was a fit using a 10-th order polynomial only achieved a mean square error of 101.87. We also found that our Ridge model using the optimal parameters reproduced the major features of the terrain in a 3D plot.

[1] R. T. Hastie and J.Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics* (Springer, New York, 2009) pp. 68–73.
[2] "Github repository with code and results: https://github.com/KarlHenrik/FYS-STK-Gruppe/tree/master/Project1," (10.10.2020).
[3] "Oppgavetekst: https://compphysics.github.io/MachineLearning/doc/Projects/2020/Project1/html/Project1.html," (10.10.2020).